applied sciences

MDPI

Article

Employing Vertical Elasticity for Efficient Big Data Processing in Container-Based Cloud Environments

Jin-young Choi [1], Minkyoung Cho [2] and Jik-Soo Kim [2,*]

[1] Gabia Inc., Seongnam 13494, Korea; ccjy2181@naver.com
[2] Department of Computer Engineering, Myongji University, Yongin 17058, Korea; mkcho@mju.ac.kr
* Correspondence: jiksoo@mju.ac.kr

Abstract: Recently, "Big Data" platform technologies have become crucial for distributed processing of diverse unstructured or semi-structured data as the amount of data generated increases rapidly. In order to effectively manage these Big Data, Cloud Computing has been playing an important role by providing scalable data storage and computing resources for competitive and economical Big Data processing. Accordingly, server virtualization technologies that are the cornerstone of Cloud Computing have attracted a lot of research interests. However, conventional hypervisor-based virtualization can cause performance degradation problems due to its heavily loaded guest operating systems and rigid resource allocations. On the other hand, container-based virtualization technology can provide the same level of service faster with a lightweight capacity by effectively eliminating the guest OS layers. In addition, container-based virtualization enables efficient cloud resource management by dynamically adjusting the allocated computing resources (e.g., CPU and memory) during the runtime through "Vertical Elasticity". In this paper, we present our practice and experience of employing an adaptive resource utilization scheme for Big Data workloads in container-based cloud environments by leveraging the vertical elasticity of Docker, a representative container-based virtualization technique. We perform extensive experiments running several Big Data workloads on representative Big Data platforms: Apache Hadoop and Spark. During the workload executions, our adaptive resource utilization scheme periodically monitors the resource usage patterns of running containers and dynamically adjusts allocated computing resources that could result in substantial improvements in the overall system throughput.

Keywords: big data; cloud computing; container; docker; resource management; virtualization; vertical elasticity

Citation: Choi, J.-y.; Cho, M.; Kim, J.-S. Employing Vertical Elasticity for Efficient Big Data Processing in Container-Based Cloud Environments. *Appl. Sci.* **2021**, *11*, 6200. https://doi.org/10.3390/app11136200

Academic Editors: Fabrizio Marozzo and Loris Belcastro

Received: 26 May 2021
Accepted: 2 July 2021
Published: 4 July 2021

1. Introduction

With the development of IT technologies, the amount of data generated is increasing at a tremendous rate ranging from TB to PB (or even more). In addition, they are coming not only from conventional structured data as in the relational database management system (RDBMS) but also from diverse unstructured or semi-structured data sources. Therefore, "Big Data" platform technologies such as Apache Hadoop [1,2] and Spark [3,4] have become crucial for distributed processing of these challenging datasets. On the other hand, "Cloud Computing" [5] has been playing an important role in providing scalable data storage and computing resources for competitive and economical Big Data processing. Although Big Data technologies can improve every part of a business from providing insights for new analytical applications to augmenting traditional on-premise systems, as the overall scale of Big Data projects grows, the complexity of managing underlying infrastructure has become challenging even for enterprises [6]. Therefore, Cloud Computing has become a viable choice for tackling Big Data problems, which results in the inevitable encounter of Big Data and Cloud Computing.

Cloud Computing provides users with scalable and elastic IT resources typically backed by large-scale data centers composed of thousands of computers through *virtual-*

ization, which allows multiple operating systems and software stacks to run on a single physical platform [5]. There have been various virtualization technologies in the cloud infrastructure, and typically, *hypervisors* such as VMWare vSphere [7] or KVM [8] have been widely used. Hypervisors basically mediate access to the physical hardware presented to each guest operating system (Guest OS), i.e., a virtual machine (VM). Despite its advantages, hypervisor-based virtualization faces many challenges in terms of performance and resource utilization, especially when running Big Data workloads, mainly due to its rigid resource consumption and guest operating system layers in virtual machines [9]. Specifically, each VM running on hypervisors includes a full set of operating systems (Guest OS), which can substantially consume shared computing resources from a physical platform. In addition, since each VM is pretty much similar to a physical machine, we cannot dynamically add or remove (i.e., scale up/down) computing resources (e.g., CPU and memory) in a *running* VM. Similar to a physical machine, a running VM should be stopped before we adjust its allocated computing resources. Therefore, hypervisor-based virtualization can mainly provide "Horizontal Elasticity" where additional VMs are provisioned when an application load increases and released when the load decreases (scale out/in) as in the AWS (Amazon Web Services) Auto Scaling mechanism [10].

On the other hand, container-based virtualization such as Docker [11] or Linux containers [12] has become an alternative to the hypervisor due to its *agile* resource management capability. Basically, a container is a standard unit of software that packages up codes and its dependencies into images so that the application runs quickly and reliably from one computing environment to another. In the case of Docker, container images become containers during runtime when they run on the Docker Engine. Docker is essentially designed to run on a Linux environment (which is also the same in other container-based virtualization technologies), and Docker's containerized software can provide the same execution environments regardless of the underlying H/W and S/W infrastructure. Unlike the virtual machines (VM), a container includes only executables and dependencies *without* a full set of guest operating system so that they can provide much more lightweight resource usage patterns [9]. Additionally, containers enable us to leverage "Vertical Elasticity" that can dynamically increase or decrease (scale up/down) computing resource allocations (e.g., CPU time, cores, memory, and network bandwidth) of a *running* container [13,14]. This agile resource management capability of containers motivated us to explore the trade-offs of employing vertical elasticity for efficient resource management in container-based cloud environments.

In this paper, we present our practice and experience of employing an adaptive resource management scheme for Big Data workloads in container-based cloud environments by leveraging the vertical elasticity of Docker [11], which is an open platform for managing and running containerized applications. Specifically, we employed two representative Big Data platforms including Apache Hadoop [1] and Spark [3] and performed extensive experiments based on several Big Data workloads from the HiBench Suite [15]. The initial implementation of Apache Hadoop [1] included the MapReduce programming model [16] with an underlying Hadoop Distributed File System (HDFS) [17], and it quickly became a de facto standard platform for effectively processing and storing "Big Data" with the advent of cluster-level operating system YARN [2,18]. Apache Spark [3,4,19] is a general-purpose, high-speed cluster computing platform designed for large-scale data processing. Spark is an open-source project and designed to be from 10 to 100 times faster than Hadoop's MapReduce by effectively employing *in-memory* computing capability. HiBench [15] is a suite of Big Data benchmarks managed by Intel and widely used to evaluate different Big Data frameworks in terms of speed, throughput, and system resource utilization.

During the workload execution, our proposed scheme periodically monitors the resource usage patterns of running containers and dynamically adjusts (scale up/down) allocated computing resources. Comprehensive evaluation results show that our adaptive container resource management scheme can effectively improve overall system utilization and user response time. To summarize, the contributions of our paper can be as follows:

- Design and implementation of an adaptive resource management scheme that can employ dynamic vertical elasticity in container-based cloud environments;
- Devising fine-grained resource coordination and monitoring policies that can reflect the characteristics of Big Data workloads;
- Application of the proposed scheme and policies to representative Big Data platforms (Apache Hadoop and Spark) and workloads (HiBench Suite);
- Comprehensive evaluation results of the adaptive resource management scheme and policies for Big Data workloads.

The rest of this paper is structured as follows. In Section 2, we discuss some of the related research work, and Section 3 presents the overall system architecture and implementation details of our adaptive container resource management scheme for Big Data workloads. Section 4 demonstrates comprehensive experimental results with the lessons learned, and finally, we conclude and discuss future work in Section 5.

2. Related Work

ELASTICDOCKER [13] was the first system to employ autonomic vertical elasticity of Docker containers, and it can reduce expenses for container customers, make better resource utilization for container providers, and improve the Quality of Experience for application end-users. However, for the evaluation, it utilized Graylog and RUBiS applications that are a powerful log management and analysis platform, and a well-known Web application benchmark, respectively. Shekhar et al. [14] presented a data-driven and predictive vertical auto-scaling framework of the cloud infrastructure that controls resource allocation adaptively at runtime for different classes of co-located workloads. Similar to the ELASTICDOCKER, they utilized the CloudSuite WebSearch benchmark. Additionally, the Resource Utilization Based Auto-scaling System (RUBAS) [20] can dynamically adjust the allocation of containers running in a Kubernetes [21] cluster. RUBAS used multiple scientific benchmarks including computational financial analysis, similarity search, dense-matrix multiply, etc.

On the other hand, this paper presents an adaptive container-based cloud resource management scheme especially optimized for Big Data workloads by effectively employing the vertical elasticity of Docker containers to improve the aggregated throughput of multiple data analytics applications running on the same physical platform. Unlike existing research work on container resource management, our resource coordination and monitoring policies are devised to be more fine-grained in order to suit the Big Data workloads. In addition, by reflecting the different resource usage patterns of Big Data workloads, our effective scaling mechanisms for CPU and memory are managed separately, and several criteria are developed. Therefore, a larger amount of computing resources can be adjusted at once, which can result in substantial improvements of resource utilization and throughput, as we will demonstrate in Section 4.

As the convergence of cloud and Big Data becomes important, we believe that our practice and experience of employing vertical elasticity of containers for Big Data workloads can provide insights and a cornerstone to researchers and practitioners in this area. Indeed, the encounter of Big Data and cloud is inevitable as we can see from the Docker container-based Big Data processing system in multiple clouds [22], where the architectural design and simulated development of Docker container-based Big Data processing system in multiple clouds are discussed. Our adaptive container resource management scheme can be a complementary technology in large-scale multi-clouds environments for big data processing.

Since container-based virtualization can provide near bare-metal performance due to its lightweight software stack, the HPC (High-Performance Computing) Cloud [23] community also has been interested in exploring the use of containers [24,25] including not only conventional Docker but also specialized containers for HPC environments such as Singularity [26] and Charliecloud [27]. Therefore, it would be also interesting to investigate

how we can effectively apply adaptive container-based cloud resource management scheme to HPC workloads.

3. System Architecture and Implementation

In this section, we present the design and implementation of our adaptive resource management scheme in container-based cloud environments that can optimize resource usage patterns of Big Data workloads by automating the vertical elasticity of Docker containers.

3.1. Execution Structure

A virtual machine (VM) in the hypervisor-based virtualization must be initially allocated a fixed amount of computing resources (e.g., CPU and memory), and in order to adjust pre-configured resources, the VM should be *suspended* (as in the physical machine), which results in a lack of agile vertical elasticity. In this case, computing resources allocated with VMs in advance can be wasted when the overall system resource requirements of running applications inside the VMs are relatively low. Due to its rigid resource consumption and guest operating system layers in virtual machines, the hypervisor-based virtualization faces many challenges in terms of performance and resource utilization, especially when running Big Data workloads [9]. Unlike these VMs, Docker allows us to dynamically adjust (scale up/down) the computing resources allocated to a container during the runtime without any suspension. This vertical elasticity of container-based virtualization enables us to optimize the computing resource allocations of running containers by continuously monitoring and reflecting the actual resource usage patterns.

Figure 1 shows the overall system architecture of our proposed adaptive resource management scheme for container-based cloud infrastructure. First, the resource usage patterns (CPU/memory) of running containers are periodically monitored and delivered to the Adaptive Resource Controller. For this purpose, we leveraged the cAdvisor [28] developed by Google, which enables us to receive real-time resource usage data of Docker containers through REST APIs. cAdvisor basically provides container users with an understanding of resource usages and performance characteristics of their running containers. It is a daemon that collects, aggregates, processes, and exports information about running containers. cAdvisor is executed as a single Docker container and can obtain resource information of containers running on the host server through a simple command, and resource monitoring using a web UI is also provided through additional settings.

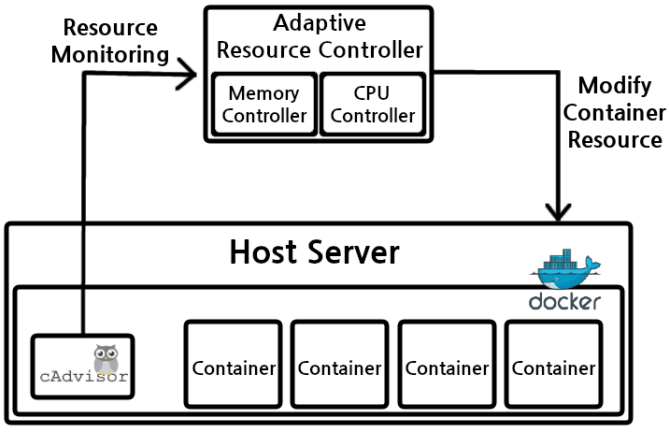


Figure 1. Adaptive resource management scheme in container-based cloud environments: the Adaptive Resource Controller periodically adjusts CPU and Memory limits of running containers if necessary, based on the monitored resource usages and pre-defined resource allocation policies.

The Adaptive Resource Controller periodically adjusts the CPU and memory limits of the running containers if necessary, based on the monitored resource usages and adaptive resource allocation policies (as described in Section 3.2). If a container's resource usage is

high, we may increase the resource limit (scale up), and if the usage is relatively low, we can decrease the allocated amounts of computing resources (scale down) so that overall computing resources can be utilized efficiently. Note that, due to the different resource usage patterns of Big Data workloads, we have employed *separate* resource allocation policies for CPU and memory, respectively. This resource management scheme is repeatedly executed at regular intervals until the workload is completed.

3.2. Adaptive Resource Allocation Policy

We collected hardware specifications of the host server to set the *maximal* resource limits of container running environments in terms of CPU and memory by using '/proc/cpuinfo' and the 'free' command, respectively. As we mentioned before in Section 3.1, for continuous resource usage monitoring, we utilized cAdvisor [28] developed by Google. However, there is another method for monitoring Docker containers by using the 'docker stats' command [11]. Compared to cAdvisor, it can be more intuitive and simpler to use; however, according to our experience, the information on container resource usages may not be accurate. In addition, sometimes, it takes relatively long time to fetch the necessary information due to the communication with containers that make it an unreliable approach for employing agile vertical elasticity.

Automating the vertical elasticity of Big Data workloads requires adaptive resource allocation based on checking container resource information at accurate intervals, so that monitoring through cAdvisor is adopted in our proposed scheme. cAdvisor can be run as a process or a Docker container, and it records resource usages of the host server and Docker containers. These collected monitoring data can be fetched through REST APIs, and the web UI for users is also provided. Some information obtained through cAdvisor are presented in Table 1.

Table 1. Information provided by cAdvisor that can be utilized for monitoring Docker containers.

Items	Description
docker/aliases	Name of docker container
spec/memory.limit	Memory limit of the docker container
docker/memory.usage	Memory usage of the docker container
stats/cpu.usage.total	CPU usage of the docker container
stats/timestamp	Monitoring time

Big Data workloads typically require large amounts of CPU and memory, and the resource usage widely fluctuates depending on the idle time when a new job is allocated or a next job is dispatched. In order to distribute and process such Big Data workloads in the cloud infrastructure, efficient resource allocation is required. For this purpose, we designed and implemented an adaptive resource allocation technology that can optimize the overall system utilization of Big Data workloads. Specifically, the monitoring cycle of Adaptive Resource Controller (in Figure 1) is set to 3 s, which can accelerate the overall resource adjustments (i.e., scaling up/down) compared to other approaches [13]. In addition, CPU and memory revision policies are devised to suit the Big Data workloads. When scaling allocated computing resources, CPU and memory are managed *separately*, and several criteria are developed according to the resource usage patterns. Therefore, a larger amount of computing resources can be adjusted at once, which can improve the aggregated system utilization and throughput.

The detailed container resource adjustment and monitoring policies that are empirically determined by reflecting the characteristics of Big Data workloads are presented in Table 2. For this purpose, we performed extensive profiling and experiments to investigate resource usage patterns (in terms of CPU and memory) of various Big Data workloads from the HiBench suite and determined the thresholds and scaling metrics presented in Table 2. Specifically, we observed that the CPU occupancy was allocated and returned by Docker containers with ease, so that thresholds for CPU coordination are set relatively

lenient. On the other hand, the allocated memory occupancy for a Docker container was not easily returned mainly due to the use of cache memory, so that thresholds for memory are made a little tighter. Therefore, in the case of memory management, it was set to be tight in order to prevent a single container from monopolizing the main memory usages since most Big Data workloads failed with insufficient memory allocations. In the case of CPU, it was set to be allocated and returned more leisurely.

Table 2. Resource coordination and monitoring policies: based on the thresholds and scaling metrics, resource allocations (CPU and memory) of running containers can be independently adjusted.

Parameter/Metric	Values
Measurement Interval	3 s
Resource Scaling Interval	3 s
CPU Scale−Up Ratios	+100%/+50%/+25% of CPU time
CPU Scale−Down Ratios	−50%/−25% of CPU time
CPU Upper Thresholds	90%/80%/70%
CPU Lower Thresholds	65%/25%
Memory Scale−Up/Down Ratios	±512 MB/±256 MB/±128 MB
Memory Upper Thresholds	95%/90%/85%
Memory Lower Thresholds	80%/50%/30%

Initially, our Adaptive Resource Controller reads in the CPU and memory specifications held by the host server. Then, it periodically (default is every 3 s) collects the list of running Docker containers along with their detailed resource usage information. If the current status of container resource usages meet the scaling conditions presented in Table 2, dynamic adjustments of computing resources for multiple running containers are performed. For example, if the current usage of CPU in a container exceeds 90% ("CPU Upper Thresholds"), we effectively scale-up the allocated CPU by adding 100% CPU time, i.e., vCPU 1 core ("CPU Scale-Up Ratios"). Similarly, if the current memory usage is less than 95% and greater than or equal to 90% ("Memory Upper Thresholds"), we add 256 MB of memory to the corresponding container ("Memory Scale-Up Ratios"). Scaling down of the allocated computing resources are performed similarly based on the lower thresholds (CPU/memory) and the scale-down ratios for CPU and memory, respectively. Note that the scaling metrics of CPU and memory are managed differently from each other (as seen from Table 2). This is because we observed that CPU usage changed rapidly every time a Big Data benchmark job is completed so that the CPU usage could decrease to the minimum usage (around 10%). Therefore, CPU scaling down policies are designed to be more coarse-grained compared to its scaling up in order to reflect rapidly changing CPU usage patterns in the Big Data workloads. Unlike the CPU, the overall memory usages during the course of Big Data workload executions have remained relatively high.

To summarize, based on our adaptive resource management scheme (in Figure 1) and resource coordination policies (in Table 2), we can effectively employ the vertical elasticity of Docker containers in a shared physical platform. For reproducibility and interested researchers and practitioners, we set up a GitHub repository for our adaptive resource management scheme in container-based cloud environments that consists of mainly eight classes and hundreds of lines of Java codes [29].

4. Evaluation

In this section, we present comprehensive evaluation results of our adaptive resource management scheme in container-based cloud environments and discuss the trade-offs in employing vertical elasticity for Big Data workloads. First, we measure the overheads of our scheme compared to the best case and perform a comparative analysis of the concurrently running Big Data workloads for both the Apache Hadoop and Spark platforms.

4.1. Experimental Setup

Table 3 shows the hardware and software configurations of our testbed, and for the experiments, we exploited HiBench Suite [15]. There are total 29 workloads in the HiBench, which are divided into six different categories including Micro, ML(Machine Learning), SQL, Graph, WebSearch, and Streaming. As representative Big Data workloads, we selected Terasort from Micro, K-means from ML, Join from SQL, and PageRank from WebSearch benchmarks. The TeraSort, K-means, and PageRank benchmarks were performed by setting the size of input data to "huge", and in the case of Join, it was set to "gigantic" due to its relatively short execution times.

Table 3. Testbed H/W and S/W configurations.

CPU	Intel(R) Xeon(R) Silver 4208 (8C 16T/2.10 GHz)
Memory	32 GB
Storage	500 GB SSD, 4TB HDD
Operating System	CentOS 7.8
Hadoop	Apache Hadoop 2.9.2
Spark	Apache Spark 2.2.3
Docker	Docker Engine 19.03.11
HiBench	HiBench 7.0

We basically conducted two types of experiments. First, in order to measure the overhead of our proposed adaptive resource allocation policy (denoted as **Adaptive**), we compared "Adaptive" with a single container where all of available computing resources of a host server are allocated, which can guarantee the *best* performance in the given computing environment (denoted as **Baseline**). In this case, Adaptive starts with a single container equipped with minimal computing resources (1 vCPU, 2.5 GB of memory) and gradually adjusts allocated resources based on the policies presented in Section 3.2. Second, we ran four different containers from the Micro, ML, SQL ,and WebSearch benchmarks in parallel and compared our proposed scheme with a fair and static resource allocation method, where each container is equally assigned 1/4 of available computing resources during the runtime (denoted as **Fair**).

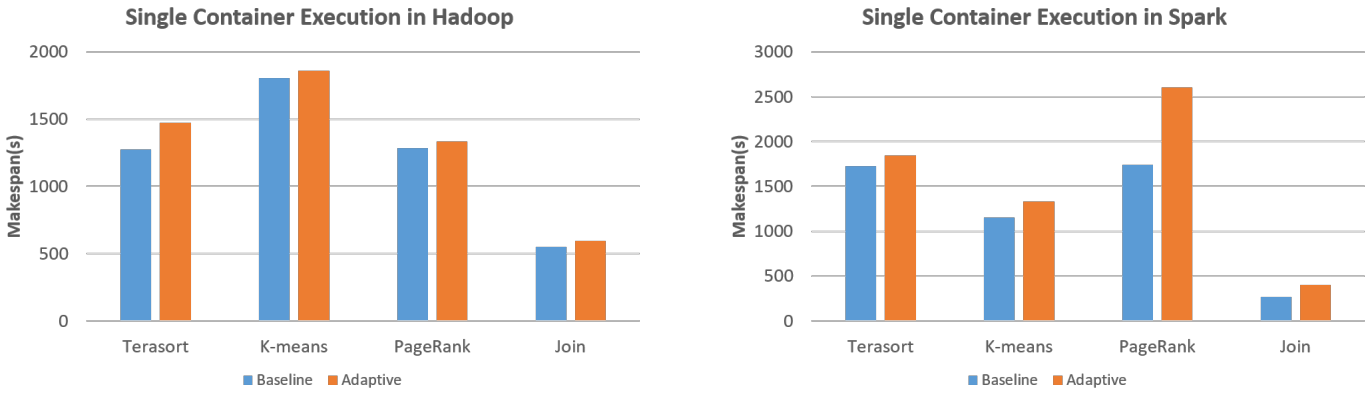
We used *makespan* (time to complete a whole job) and *resource utilization* (for CPU and Memory) as our performance metrics. Since the CPU allocations can change whenever a Big Data workload is executed, we run every workload five times and measured the average makespans.

4.2. Experimental Results

Figure 2 shows the performance evaluation results of running a single container with "Baseline" and "Adaptive" (as described in Section 4.1) for Hadoop and Spark, respectively. As we can see from the results, the overall execution times increased in the Adaptive approach. For the K-means benchmark, the execution time increased by only 3% compared to the optimal result (Baseline) when using the Hadoop resulting in the lowest variation (as seen from Figure 2a). However, in the case of PageRank benchmark in Spark, the execution time increased by almost 50% with the largest variation, as depicted in Figure 2b.

The main reason for high job execution time variance is potentially due to the difference in time-specific resource usage patterns of each Big Data workload, and the lack of initially assigned computing resources in the Adaptive approach. The initial resources of a single container in the Adaptive approach are allocated to use only one vCPU and 2.5 GB of memory, which can result in a potential bottleneck in the job execution, especially when it requires larger amounts of initial computing resources until the Adaptive can meet those requirements. Since our adaptive resource allocation scheme is based on a kind of hill-climbing algorithm that increases resource allocation values continuously until it achieves a peak solution, in the case of the single container experiment, our Adaptive approach was not fast enough to meet the initial resource requirements for some Big Data

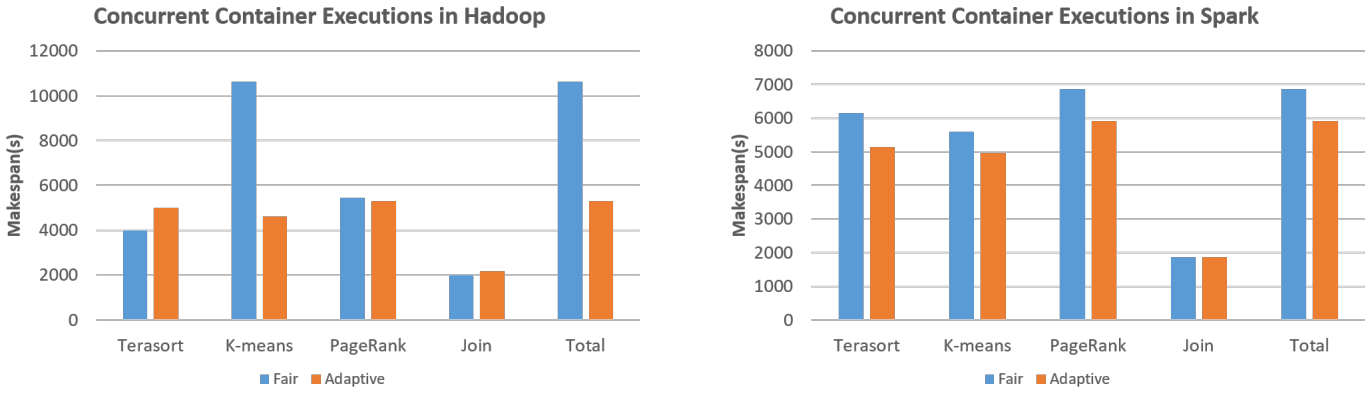
workloads. To summarize, an average of 18.7% of overhead can occur when the overall system is idle with our proposed scheme due to its scaling-up mechanism, which can be further studied in the future.



(**a**) Hadoop (**b**) Spark

Figure 2. Experimental results of single-container executions that can show the overheads of our proposed approach compared to the best performance in the given computing environment.

Figure 3 shows the performance evaluation results of parallel executions of four different containers with "Fair" and "Adaptive" (as described in Section 4.1) for Hadoop (in Figure 3a) and Spark (in Figure 3b), respectively. Although execution times in the Adaptive approach increased for some benchmarks (e.g., for the Terasort and Join benchmarks using Hadoop in Figure 3a, 25% of increase and 10% of increase, respectively), overall, our proposed scheme shows very competitive performance compared to that of Fair in terms of makespan. For example, in the K-means benchmark, the Adaptive scheme dramatically improves the makespan compared to that of Fair by 56% and 12% in Hadoop and Spark, respectively. This is because unlike the Fair approach where available computing resources are statically and equally assigned, our adaptive resource allocation scheme can effectively utilize idle computing resources during the course of job executions.



(**a**) Hadoop (**b**) Spark

Figure 3. Experimental results of concurrent container executions that can show the aggregated throughput improvements of our proposed approach.

Figures 4 and 5 show the resource utilization patterns of the "Fair" and "Adaptive" approaches in terms of main memory usages. As we can see from the results, the Fair approach cannot utilize available resources whenever a job is completed, which results in a waste of allocated computing resources and increased makespan even compared to the "Baseline". On the other hand, our proposed Adaptive approach can effectively leverage idle computing resources during the course of job executions, which can result

in improved resource utilization and makespans, as depicted in the exemplary case of K-means benchmark (Figure 5) and Figure 3.

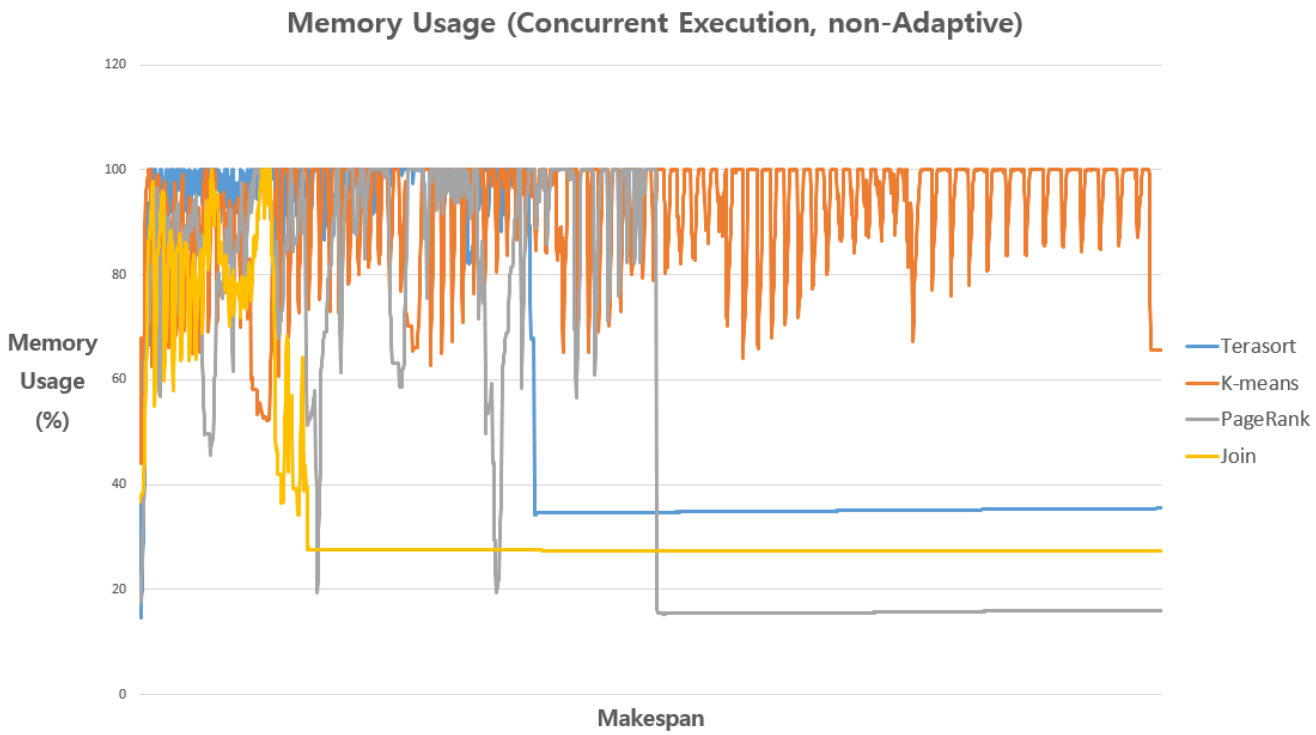


Figure 4. Memory usages of benchmarks in the Fair approach to show an exemplary case of inefficient resource usage patterns.

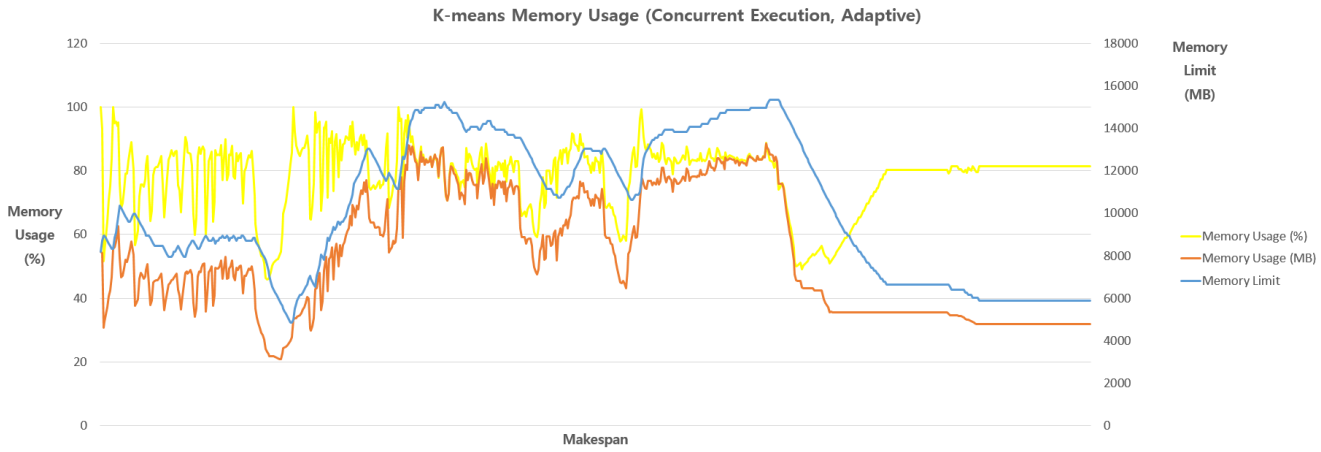


Figure 5. Memory usages in the K-means benchmark in the Adaptive approach to show an exemplary case of efficient resource usage patterns.

In the case of applications where the overall execution time increased compared to the Fair approach (e.g., Terasort and Join in Hadoop as seen from Figure 3a), the workload is typically divided into several execution steps so that, between those steps, other running containers may have acquired idle computing resources. This can be an interesting trade-offs of making the monitoring and adjusting cycles short (in our case, 3 s, as discussed in Section 3.2). If there are idle cycles longer than our predefined monitoring period between multiple steps of a Big Data job, allocated computing resources can be redistributed to other containers even though that Big Data job has not been completely finished yet. Nevertheless,

based on the effective vertical elasticity management of Docker containers, our proposed scheme can achieve aggregated performance improvements by 50.1% and 13.7% in Hadoop and Spark, respectively (as seen from "Total" in Figure 3a,b) for representative Big Data workloads. Tables 4 and 5 show the summarized experimental results for single-container executions and concurrent container executions, respectively.

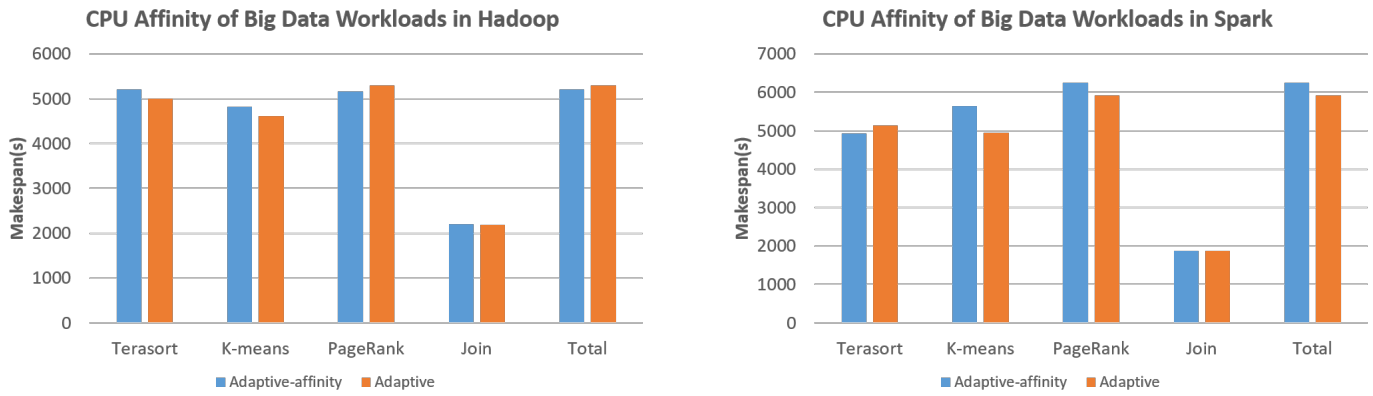
Table 4. Summary of single-container executions (sec).

Benchmark	Hadoop (Baseline)	Hadoop (Adaptive)	Spark (Baseline)	Spark (Adaptive)
Terasort	1275.2	1471.2	1725.6	1848.8
K-means	1806.2	1860.4	1151.6	1333
PageRank	1285.8	1334	1741.8	2607.6
Join	552.4	595.8	270.8	398.8

Table 5. Summary of concurrent container executions (sec).

Benchmark	Hadoop (Fair)	Hadoop (Adaptive)	Spark (Fair)	Spark (Adaptive)
Terasort	3992	5007.6	6146.4	5140.8
K-means	10,621.2	4620.4	5605.4	4953.2
PageRank	5432.8	5301	6860.4	5918.4
Join	2002.8	2187.8	1864.6	1868.6
Total	**10,621.2**	**5301**	**6860.4**	**5918.4**

Docker not only can limit the maximum CPU usage of a container based on CPU time but also supports the method of limiting the use of only specific CPUs by setting a fixed set of CPU thread numbers (called "CPU Affinity") [30]. In the case of conventional CPU time-based approach, since the CPUs being used can continuously change, the overhead of context switching may occur. Therefore, in order to investigate the effects of CPU pinning (i.e., setting the CPU affinity) on Big Data workloads, we performed other experiments, as seen in Figure 6. We compared our basic adaptive resource allocation scheme (**Adaptive**) with the modified CPU allocation method from time sharing to affinity (**Adaptive-affinity**).



(**a**) Hadoop (**b**) Spark

Figure 6. Effects of CPU Affinity, which limits the use of only specific CPUs by setting a fixed set of CPU thread numbers for Big Data workloads.

Overall, the effects of applying CPU affinity to the resource allocation of containers seem not so significant considering that we are running Big Data workloads, which are not typically compute-intensive. For example, the aggregated makespan of Adaptive affinity is improved only by 1.9% compared to the basic Adaptive approach in the Hadoop platform (as seen from Figure 6a). Interestingly, the aggregated makespan of Adaptive affinity becomes worse by 5.7% compared to the original approach in the Spark platform, as seen

from Figure 6b. This can be potentially due to coarse-grained CPU allocations compared to the time-sharing method, especially when CPU operations may be required for calculating RDDs [19] for iterative workloads such as K-means and PageRank in the Apache Spark platform. Therefore, we plan to investigate the application of various resource constraints in the Big Data workloads as one of our future works.

5. Conclusions and Future Work

In this paper, we presented our practice and experience of employing an adaptive cloud resource utilization scheme based on the container-based vertical elasticity especially optimized for Big Data workloads. We utilized two representative Big Data platforms, Apache Hadoop and Spark, and performed comprehensive experiments based on several Big Data workloads from the HiBench Suite. During the workload execution, our proposed scheme periodically monitors the resource usage patterns of running containers and dynamically adjusts (scale up/down) allocated computing resources. Performance evaluation results show that idle computing resources can be efficiently utilized, which can result in up to 50.1% and 13.7% of aggregated throughput improvements for Hadoop and Spark, respectively.

Our approach is simple yet effective so that, as the convergence of cloud and Big Data becomes important, we believe that our experience of employing vertical elasticity of containers for Big Data workloads can provide useful insights and a cornerstone in terms of how to start applying vertical elasticity of Docker containers for Big Data workloads, considerable parameters and metrics for elastic resource managements, and an implementation strategy for building a complete monitoring and adjusting system for running containers to researchers and practitioners. However, our resource coordination and monitoring policies are empirically determined by investigating the resource usage patterns of Big Data workloads and are applied statically during the course of job executions. This strategy can limit the effectiveness of the proposed scheme since monitoring intervals, thresholds, and scaling metrics are configurable parameters that may not be universally applied to a wider range of Big Data applications. In addition, the performance improvements in the Spark platform were relatively smaller compared to that of Hadoop possibly due to Spark's in-memory computing capability, which means that we need more sophisticated container resource management policies.

Therefore, our future work includes devising a dynamic parameter configuration methodology in order to reflect the variety of Big Data application executions (e.g., Graph and Steaming, SparkSQL, Spark Mllib, etc.), reducing the overhead of scaling mechanisms, and application of various resource constraints in the Big Data workloads (as we discussed in Section 4.2).

Author Contributions: Conceptualization, J.-y.C. and J.-S.K.; methodology, J.-y.C. and J.-S.K.; software, J.-y.C.; validation, J.-y.C., M.C., and J.-S.K.; formal analysis, M.C. and J.-S.K.; investigation, J.-y.C.; resources, J.-y.C.; data curation, J.-y.C.; writing—original draft preparation, M.C. and J.-S.K.; writing—review and editing, M.C. and J.-S.K.; visualization, M.C. and J.-S.K.; supervision, J.-S.K.; project administration, J.-S.K.; funding acquisition, J.-S.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.NRF-2019R1A2C1005360) and the Ministry of Education (NRF-2020S1A3A2A02103899).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. The Apache Hadoop Project: Open-Source Software for Reliable, Scalable, Distributed Computing. Available online: http://hadoop.apache.org/ (accessed on 3 July 2021).
2. Vavilapalli, V.K.; Murthy, A.C.; Douglas, C.; Agarwal, S.; Konar, M.; Evans, R.; Graves, T.; Lowe, J.; Shah, H.; Seth, S.; et al. Apache Hadoop YARN: Yet Another Resource Negotiator. In Proceedings of the 4th Annual Symposium on Cloud Computing (SoCC'13), Santa Clara, CA, USA, 1–3 October 2013.
3. Apache Spark: A Unified Analytics Engine for Large-Scale Data Processing. Available online: https://spark.apache.org/ (accessed on 3 July 2021).
4. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster Computing with Working Sets. In Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud 2010), Boston, MA, USA, 22 June 2010.
5. Buyya, R.; Broberg, J.; Goscinski, A. *Cloud Computing Principles and Paradigms*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2011.
6. Thusoo, A. Why the Cloud and Big Data? Why Now? 2016. Available online: https://www.datanami.com/2016/12/16/cloud-big-data-now/ (accessed on 3 July 2021).
7. vSphere Hypervisor. Available online: https://www.vmware.com/products/vsphere-hypervisor.html (accessed on 3 July 2021).
8. Kivity, A.; Kamay, Y.; Laor, D.; Lublin, U.; Liguori, A. KVM: the Linux Virtual Machine Monitor. In Proceedings of the 2007 Ottawa Linux Symposium (OLS'-07), Ottawa, ON, Canada, 27–30 June 2007.
9. Zhang, Q.; Liu, L.; Pu, C.; Dou, Q.; Wu, L.; Zhou, W. A Comparative Study of Containers and Virtual Machines in Big Data Environment. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 178–185.
10. AWS Auto Scaling. Available online: https://aws.amazon.com/autoscaling/ (accessed on 3 July 2021).
11. Docker: Empowering App Development for Developers. Available online: https://www.docker.com/ (accessed on 3 July 2021).
12. Linux Containers: Infrastructure for Container Projects. Available online: https://linuxcontainers.org/ (accessed on 3 July 2021)
13. Al-Dhuraibi, Y.; Paraiso, F.; Djarallah, N.; Merle, P. Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER. In Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, HI, USA, 25–30 June 2017; pp. 472–479.
14. Shekhar, S.; Abdel-Aziz, H.; Bhattacharjee, A.; Gokhale, A.; Koutsoukos, X. Performance Interference-Aware Vertical Elasticity for Cloud-Hosted Latency-Sensitive Applications. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 82–89.
15. HiBench Suite: The Bigdata Micro Benchmark Suite. Available online: https://github.com/Intel-bigdata/HiBench (accessed on 3 July 2021).
16. Dean, J.; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
17. Shvachko, K.; Kuang, H.; Radia, S.; Chansler, R. The Hadoop Distributed File System. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'10), Incline Village, NV, USA, 3–7 May 2010.
18. Murthy, A.; Vavilapalli, V.; Eadline, D.; Niemiec, J.; Markham, J. *Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2*; Addison-Wesley Data & Analytics: Boston, MA, USA, 2014.
19. Zaharia, M.; Chowdhury, M.; Das, T.; Dave, A.; Ma, J.; McCauley, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12), San Jose, CA, USA, 25–27 April 2012.
20. Rattihalli, G.; Govindaraju, M.; Lu, H.; Tiwari, D. Exploring Potential for Non-Disruptive Vertical Auto Scaling and Resource Estimation in Kubernetes. In Proceedings of the 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), Milan, Italy, 8–13 July 2019; pp. 33–40.
21. Kubernetes: Production-Grade Container Orchestration. Available online: https://kubernetes.io/ (accessed on 3 July 2021).
22. Naik, N. Docker container-based big data processing system in multiple clouds for everyone. In Proceedings of the 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, 11–13 October 2017.
23. Kotas, C.; Naughton, T.; Imam, N. A Comparison of Amazon Web Services and Microsoft Azure Cloud Platforms for High Performance Computing. In Proceedings of the 2018 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 12–14 January 2018.
24. Arango, C.; Dernat, R.; Sanabria, J. Performance Evaluation of Container-based Virtualization for High Performance Computing Environments. *arXiv* **2017**, arXiv:1709.10140.
25. Abraham, S.; Paul, A.K.; Khan, R.I.S.; Butt, A.R. On the Use of Containers in High Performance Computing Environments. In Proceedings of the 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), Beijing, China, 19–23 October 2020.
26. Kurtzer, G.M.; Sochat, V.; Bauer, M.W. Singularity: Scientific containers for mobility of compute. *PLoS ONE* **2017**, *12*, e0177459. [CrossRef]
27. Priedhorsky, R.; Randles, T. Charliecloud: unprivileged containers for user-defined software stacks in HPC. In Proceedings of the 2017 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, CO, USA, 12–17 November 2017.
28. Google cAdvisor. Available online: https://github.com/google/cadvisor (accessed on 3 July 2021).

29. Adaptive Resource Management Scheme in Container-Based Cloud Environments. Available online: https://github.com/IDPL-MJU/ContainerVerticalElasticity/ (accessed on 3 July 2021).
30. Runtime Options with Memory, CPUs, and GPUs. Available online: https://docs.docker.com/config/containers/resource_constraints/ (accessed on 3 July 2021).